Issue #3
September 1984

Ye Zee News

Registered By Australian Post
Publication No. SBG0711

Return Address
30 Seventh ave. Cheltenham S.A. 5014

This months issue has several contributions from readers, I hope you find them as interesting as I did.

There is another of Ross Constantine's tutorial programs. In his own words Ross views the program as a starting point for those interested in education type programs rather than the end product.

A method of preventing double character entry when using INKEY$ and some notes on INTEGER and STRING storage are also included.

There are two contributions from Ron Carson. The first is a program for logging callsigns for amateur radio buffs. The other is machine code tutorial kindly written for Ve Zee News. Ron worked "blind" on this in as much as he doesn't know what level people need to start at. A bit of feed back will help him taylor future tutorials. He hopes these will be an on going thing, the 2nd and 3rd are alredy written.

John

CONTRIBUTIONS

The following points will help those thinking of sending programs for publication in Ve Zee News.

- If possible send a listing and a tape of your program. This helps me prepare the newsletter and the tape will be returned when I have finished.
- Include some notes on what the program does and how to use it. What is painfully obvious to the author is often obscure to we users.
- For long or complex programs let me know if I can supply readers with a copy of it on tape or if you are prepared to do this yourself.
- If you have software for sale, let me know. A brief description of your software and its price and availability will be included in Ve Zee News at no charge (for subscribers).

Some notes on DEFINT & DEFDBL

Using DEFINT A-C (for example) defines all variables starting with A B or C as integer without the need to use the % suffix.

i.e. DEFINT A : AB=1

is the same as

AB%=1

DEFDBL is used in the same way but defines variables as double precision (that is calculations are done to 16 figure accuracy). These variables must be used with care in arithmetic expressions. All variables within an expression must be double precision type or the result will be inaccurate. The following printout shows this.

```
READY
1 ?A-B
POKE31469,155
READY
5 D=2:E=3
10 A1=D/E
15 B1=2:B2=3
20 A2=B1/B2
25 PRINTA1:PRINTA2


RUN
 .6666666865348816
 .6666666666666667
```

The first result, using single precision (normal) variables is clearly not correct. However the second result using all double precision variables is correct.

LOTTO SELECTOR

It seems that if your VZ 200 has a BASIC version other than version 2.0 the display for last months lotto program will be inversed. Try the following changes to the program and see if that helps.

All data in lines 115 to 140 should have 64 subtracted from each number. Then lines 235, 240, 335 and 340 should be as follows.

```
235 IFA1>64THEN215ELSEA1=A1+64
240 A2=PEEK(PO+1)+64
335 IFPEEK(NN)<64THEN355
340 A1=PEEK(NN)-64:A2=PEEK(NN+1)-64
```

Variable Storage

In RAM immediatly above the BASIC program is an area called the Variable Listing Table (VLT). A pointer to the start of the VLT is stored at 30969 &30970. When a BASIC program is running, as a variable is used the VLT is searched and if the variable has already been used it is found, if it has not been used then an area of RAM is assigned to it and labeled. The variables are stored in the order in which they are used.

Before the program is RUN no variables are stored, this explains why some long programs give an OUT OF MEMORY error only after they start running.

The program below displays each variable as it is stored. You can study how variables are stored by adding lines like the following to the program.

```
100 RV=1.00
110 RW%=2
120 RX$="AFTER"
```

The top of the middle column below is a screen COPY of the input used to generate the output listed in the right column. The output from the program gives the address, the byte value and the meaning or binary value (if the byte is part of the variable value).

The first byte of each variable storage area contains a number specifying how many bytes contain the variable value. 2 for integer, 3 for string, 4 for single precision and 8 for double precision.

The next two bytes contain the variable name (in reverse order), this explains why the VZ 200 only looks at the first two characters of a variable name.

The remaining bytes have different meanings for each variable type. These meanings are explained below.

```
PROGRAM TO DISPLAY VLT REGION


1000 DIM TY$(6)
1010 TY$(0)="INTEGER ":TY$(5)="VALUE"
1020 TY$(1)="STRING "
1030 TY$(2)="SINGLE PRECISION "
1040 TY$(3)="VARIABLE":TY$(4)="NAME "
1050 TY$(6)="DOUBLE PRECISION "
1060 AD=256*PEEK(30970)+PEEK(30969)
1070 GOSUB 1220
1080 NB=PEEK(A1)
1090 PRINTAD;" ";NB;TY$(NB-2)
1100 AD=AD+1 : GOSUB 1220
1110 PRINTAD;" ";CHR$(PEEK(A1));"    )";
TY$(3)
1120 AD=AD+1 : GOSUB 1220
1130 PRINTAD;" ";CHR$(PEEK(A1));"    )"
";TY$(4)
1140 AD=AD+1 : GOSUB 1220
1150 BY=PEEK(A1) : GOSUB 2000
1160 PRINTAD;BY;" ";A$
1170 NB=NB-1
1180 IF NB>0 THEN 1140
1190 AD=AD+1
1200 INPUT"PRESS RETURN TO CONTINUE";A$
1210 GOTO 1070
1220 IFAD>32767THENA1=AD-65536 ELSEA1=AD
1230 RETURN
2000 A$=""
2010 FOR BI=7TO0 STEP-1
2020 BV=2^BI : RC=BY AND BV
2030 IF RC>0THENA$=A$+"1" ELSE A$=A$+"0"
2040 NEXT BI : RETURN
```

```
READY
1 #Q
POKE31469,155
READY
2 QW=1.23456789012345
10 ZX%=126
20 XY$="AS YOU WISH"
```

The output from this Program will be a bit more tidy if line 1160 is changed as follows,

```
1160 PRINTAD;USING###,BY;" "
;A$
```

Next month the storage of single precision and double precision variables will be looked at.

The storage of arrays will be delt with at a later date, as this is a whole topic in itself.

```
32192   8 DOUBLE PRECISION
32193  W    )VARIABLE
32194  Q    )  NAME
32195  232  11101000
32196  206  11001110
32197  98  01100010
32198  20  00010100
32199  82  01010010
32200  6  00000110
32201  30  00011110
32202  129  10000001
PRESS RETURN TO CONTINUE?

32203   2 INTEGER
32204  X    )VARIABLE
32205  Z    )  NAME
32206  126  01111110
32207  0  00000000
PRESS RETURN TO CONTINUE?


32208   3 STRING
32209  Y    )VARIABLE
32210  X    )  NAME
32211  11  00001011
32212  29  00011101
32213  123  01111011
PRESS RETURN TO CONTINUE?


32214   4 SINGLE PRECISION
32215  D    )VARIABLE
32216  A    )  NAME
32217  0  00000000
32218  180  10110100
32219  123  01111011
32220  143  10001111
PRESS RETURN TO CONTINUE?
```

© John Waters September 1984
All material in Ve Zee News is subject to copyright. Contributed programs are reproduced with permission and on the understanding that they are for the private use of subscribers only. Copyright is retained by the author.

The last two bytes of an integer contain the binary value of the variable. For positive numbers the storage is simply as binary so the value is,

256*2nd byte + 1st byte ie 126 for ZX% in our example printout.

For negative numbers it is not that simple, put one in and try it. The reason for this is that negative numbers are stored as two's compliment, The two's compliment of a binary number is found by changing all 0's to 1's and all 1's to 0's then adding 1 and ignoring any carry bit.

+126 is	01111110	binary
compliment this is	10000001	
add 1 gives	10000010	this is -126
compliment again	01111101	
add 1	01111110	this is +126 again

A simple way to do this is to do the following if the first bit of the last byte is 1

value = (256*2nd byte + 1st byte)-65536

Fortunately our trusty BASIC interpreter takes care of all this housekeeping for us.

STRING

The 4th. byte of a string variable is the length (in characters) of the string. For example, 11 for the string "AS YOU WISH". The next two bytes contain the address of the first character of the string, ie 123*256+28 (31517) in the example below. Note that for a string assigned a value in the program (eg A$="QWERTY") the address will be in the program storage list and will correspond to the address of the character in the program line.

eg 10 A$="ABCDEF"

the address will be this point in RAM. If the string is the result of an input or concatenation (eg A$=C$+D$) it will be stored in the top of the RAM above the address found by the following statement.

```
PRINT 256*PEEK(30881)+PEEK(30880)
```

This top end of RAM is set side for string storage. Either 50 bytes by the ROM start up routine or to the value set by a CLEAR command.

While writing a spelling/typing tutor program I ran into trouble using the INKEY$ command.

My first attempt at inputing characters was like this

```
100 A$ =INKEY$:IFA$ =""THEN 100
110 PRINTA$;:GOTO100
```

However I found that the keyboard scanning rate was such that on pressing a key a series of about 10 characters was printed.

Several other attempts were tried using FOR-NEXT loops to effectively slow the scan rate. These attempts were not particularly successfull.

With a bit more thought I have come up with the coding below...

```
100 A$=""
110 T$=INKEY$:IF T$="" AND A$="" THEN100
120 IFT$="" THEN 130 ELSE A$=T$:GOTO 110
130 PRINT A$;:GOTO 100
```

The logic behind this is as follows...

Two string stores are used. T$ which is the current INKEY$ and A$ which is the previous INKEY$. The coding will only print a character when the key is released. ie T$="" and A$= to the key that was pressed. Note a character will not be printed on pressing a key.

I have found that occasionally a key bounce will give a double character The following modifications will check A$ against a previously stored array

```
90 C=0
130 C=C+1: IFA$=Q$(C)THEN140
135 A$="":C=C-1:SOUND 27,2
140 PRINT A$;:IF C<[array length] THEN 100
```

Q$(C) is an element of the previously stored array.

It is left to the reader to use the routines above as a section of their own program.

Mark Shirley

More on Invisible Commands

A number of readers have asked about the use of the extra TRS 80 commands without using the utility in issue 1. It is possible and often more convenient to POKE the tokens directly into your program instead of going through a utility.

This is most easily done if the line to be changed is the first (ie lowest line number) line of the program. The first character or token of the first line is stored at 31469 in RAM so within the line it is easy to keep track of where to POKE the token.

The following are some examples.

```
READY
1 PRINT?(32,"*")
POKE31470,196
READY
RUN
********************************
READY

READY
1 PRINT?(X$)
POKE31470,218
READY
RUN
 50
READY

1 ?A-C
POKE31469,155
READY
2 A=2:B=3:C=A/B
4 PRINT C
RUN
 .666666666666667
READY
```

Any line number can be entered in this way as long as it is the lowest line number at the time of entering. For example if your program is as follows,

```
100 REM DEMO PROGRAM
 ⋮
120 RANDOM
 ⋮
1010 PRINT M
 ⋮
2000 END

ENTER LINE 1010 FIRST AS FOLLOWS,
1010 PRINT ?
POKE31471,200
READY

THEN LINE 120

120 ?
POKE31469,134
READY
```

Then enter the rest of the lines as normal.

Beware When a line with a TRS 80 token is edited the contents are lost

COMMUNICATIONS REGION

30882 & 83	current line number
30887 & 88	address of the start of the keyboard buffer
30934 & 35	address of next available location in string area
30938 & 39	line number of last DATA statement read
30965 & 66	last line number executed
30973 & 74	the starting address of the free space in RAM
30977	variable default type

More on Variable Type Definition

The area in the communications region from 30977 to 31022 is a table of the default types for variables. There are 26 bytes in the table, one for each letter of the alphabet, 30977 for A to 31022 for Z.

On start up the ROM sets all these bytes to 4 (ie all variables have a default type of single precision). The default type is the type assigned to a variable by the computer if you do not use a type suffix (% or $) or DEFINT or DEFDBL command.

This information gives us an easy way to define variable types. That is to POKE the correct value into this table. The values are 2 for integer, 4 for single precision 8 for double precision and 3 for string.

For example,

```
POKE 30977,2
```

defines variables starting with A as integer.

```
FOR K=13 TO 20
POKE 30977+K,8
NEXT K
```

defines variables starting with N,O,P,Q,R,S,T or U as double precision type.

```
170 CLEAR 2000
180 DIM C1$(2000)
190 CLS
200 REM
210 CLS:PRINT:PRINT" NEXT CALL SIGN, SEE BELOW":PRINT
211 PRINT:PRINTTAB(3);" LIST :- LIST WITHOUT SORT"
212 PRINT:PRINTTAB(3);" SORT :- SORT CALL SIGNS"
213 PRINT:PRINTTAB(3);" PRINT:- LIST ON PRINTER"
214 PRINT:PRINTTAB(3);" END  :- END PROG."
215 PRINT:PRINTTAB(3);"      :- ENTER CALLSIGN"
216 PRINT:PRINT" ENTER :- ";:INPUT A1$
220 IF A1$="SORT" THEN 500
230 IF A1$="LIST" THEN 700
235 IF A1$="END" THEN CLS:END
236 IF A1$="PRINT" THEN 950
240 FOR I=1TO LEN(A1$)
245 NEXT I
260 CLS
270 REM
280 FOR I=1 TO N
290 IF A1$=C1$(I) THEN 400
300 NEXT I
310 REM
320 N=N+1
325 C1$(N)=A1$
330 PRINT:PRINT TAB(3)" ";A1$;" IS NEW CALL SIGN"
340 PRINT:PRINT:PRINT TAB(5)" ";N;" CALLS LOGGED "
345 FOR X = 1 TO 1000
350 NEXT X
360 GOTO 200
400 REM
410 PRINT:PRINT:PRINT TAB(4)" ";A1$;" ALREADY LOGGED"
420 GOTO 340
500 REM
510 CLS:PRINT:PRINT TAB(12);"SORTING":PRINT
520 FOR I=1 TO N
530 A1$=C1$(I)
540 PRINT "*";
550 FOR J=I TO N
560 IF A1$<=C1$(J) THEN 580
570 B1$=C1$(J)
572 C1$(J)=A1$
574 A1$=B1$
580 NEXT J
590 C1$(I)=A1$
600 NEXT I
610 PRINT:PRINT:PRINTTAB(9);"SORT COMPLETE"
620 PRINT:PRINT:PRINT:PRINT" DO YOU WANT A PRINTOUT?":PRINT
621 PRINT"  [PRINT] = PRINTOUT TO PRINTER"
622 PRINT"  [YES]   = PRINTOUT TO VDU"
623 PRINT"  [NO]    = RETURN TO MENU"
625 K$=INKEY$
626 I$=INKEY$:IFI$=""THEN625
627 IFI$<>"Y"ANDI$<>"P"ANDI$<>"N"THEN625
630 IF I$="N" THEN 190
635 IF I$="P" THEN 950
640 IF I$="Y" THEN 700
700 PRINT
702 CLS:PRINT:PRINT TAB(7);"CALL SIGNS LOGGED":PRINT
710 FOR I=1 TO N
750 PRINT C1$(I),
760 NEXT I
764 PRINT:PRINT"  PRESS >>SPACE<< TO CONTINUE"
765 K$=INKEY$
770 I$=INKEY$:IFI$<>" "THEN765
780 GOTO 900
900 REM
910 CLS:PRINT:PRINT:PRINT:PRINT"   DO YOU WANT TO STOP NOW?"
912 PRINT:PRINT"         Y=YES      N=NO "
913 PRINT:PRINT:PRINT:PRINT:PRINT TAB(5)" ";N;" CALLS LOGGED "
915 K$=INKEY$
917 X$=INKEY$:IF X$=""THEN 917
920 IF X$="Y" THEN CLS:END
925 IF X$="N" THEN 200
927 IFX$<>"Y"ANDX$<>"N"THEN917
950 LPRINT TAB(15);"CALL SIGNS LOGGED"
955 LPRINT
960 FOR I=1 TO N+1
970 LPRINT C1$(I),
980 NEXT I
990 GOTO 900
```

MACHINE CODE - ASSEMBLER TUTORIAL

This is the first in a series (I hope), to look at the machine code and Assembler language of the Z80 chip, especially as it relates to the VZ-200. There will surely be occasional errors but we may all learn a little, and it does seem (from your questions), that many want help in Assembler code. SOON I hope some of you will assist me with short, illustrative and usefull routines. Anyway we will see how it goes!!!!

It will be most useful for you to have an editor/assembler and disssembler, and at present, I realise these are few and far between. Although there are a few aroud, and D.S. has them on the way.

THE REGISTERS.

In a sense it is a pity to have to start here for to see Registers in action easily, a disassembler with a "Trace" function lets us see immediately what is happening.
The registers are for data transfer, so are used all the time in programming. Often we load a register with a number, alter it in some way or other, and having stored it, place it in a known position in memory for future use.

The registers of the Z80 are named by letters: one letter for 8 bit registers, and two letters for 16 bit (or double byte) registers.

There are two sets of the main registers, the alternate being only worked on following 'register exchange' instructions. We will keep away from this group for the time being.

As will be detailed in the next tutorial, the A register is the busiest, is called the accumulator and is a single byte (or 8 bit) register. It's valuable because the accumulator can be loaded from any other register so it tends to be the supermarket trolley carrying material to and fro.

Coupled 'lightly' with the A register is the F or flag register. It contains six status bits in the following order:.....

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT1	BIT 0
[S]	[Z]	[ ]	[AC]	[ ]	[P/O]	[N]	[C]
sign	zero	unused	auxiliary carry		Parity/ o'flow	sub/ tract	carry

What could be more confusing? Well I had to tell you but at this time we will leave the Flags and make them a subject on their own!

THE GENERAL PURPOSE REGISTER PAIRS

These are three in number (ignoring the alternate registers). They are :- BC DE HL
Each 8 bit part can be used as a single register in its own right.
B C D E H L
These six 8 bit registers have a very limited number of accumulator-like operations, and in practice, they are used to store data bytes for use later in the accumulator (or A register), for the data is easily and quickly got in and out of them through A register.

Having mentioned about storing twice, it may disappoint you to know there is no store command in the Z80 chip! HOW do we cope?

DATA TRANSFER OPERATIONS

Most of this is done by using LD (load) instructions. Because this is through registers (see above), such data may be transfered in groups of 8 or 16 bits (1 or 2 bytes).

Other instructions like EX, EXX, LDIR and LDI (for blocks), PUSH and POP can be used to move, and store data.

But lets stay with the LD instruction.
The LD instruction always has two operands. The first is the location were the data is stored (it may be a register or memory address), and is called the 'destination'.

The second is the original location of the data to be transferred, this is the 'source'. Whilst it may be a register, you'll soon run out of these and quite often it is a memory location that you used earlier as a 'store'.

What are the possibilities then?

register <- register
e.g. LD A,B

register <- memory
LD A,(HL)

register <- immediate data
e.g. LD A,25H or LD HL,7120H

memory <- register
e.g. LD (HL),A or PUSH BC

memory <- memory
LDI or LDIR

memory <- immediate data
e.g. LD (HL),54H

Now you should be able to write an assembly language programme to set all the 8 bit register contents to, say:
A to 1, B to 0FH, C to 2, D to 0EH, E to 3, H to 0CH, L to 4

By loking up a book on Z80 code, you could then find the Hex code for each of these instructions, (the values to be inserted you know already), and manually write the Machine code, then G<address> the routine (if you are lucky enough to own a monitor). Unfortunately, while it would work, you couldn't see any action without a disassembler because the registers can't be examined.

The D.s. disassembler though in Basic would allow you to see the action and check registers.

At present this part of your newsletter may inform you and give you an understanding of machine code, don't put tis where you can't find it because machine code utility programmes are poping up around the place, and when you get yours you can get these tutorials out again and go over and over each one until you understand what this new language is all about.

SEE YOU NEXT MONTH.

Ron Carson

GEOGRAPHY

```
5 DIMA$(30)
10 LETA$(1)="AUSTRALIA"
15 LETA$(2)="NORTHERN"
16 LETA$(3)="HOBART"
17 LETA$(4)="ADELAIDE"
18 LETA$(5)="OCEAN"
19 LETA$(6)="PACIFIC"
20 LETA$(7)="WESTERN"
21 LETA$(8)="INDIAN"
22 LETA$(9)="TASMAN"
23 LETA$(10)="SEA"
24 LETA$(11)="TIMOR"
25 LETA$(12)="TERRITORY"
26 LETA$(27)="SOUTHERN"
27 LETA$(28)="AUST"
30 LETA$(13)="QUEENS"
35 LETA$(14)="A.C.T."
40 LETA$(15)="LAND"
45 LETA$(16)="PERTH"
50 LETA$(17)="NEW"
55 LETA$(18)="BRISBANE"
60 LETA$(19)="SOUTH"
65 LETA$(20)="SYDNEY"
70 LETA$(21)="WALES"
75 LETA$(22)="MELBOURNE"
80 LETA$(23)="VICTORIA"
85 LETA$(24)="CANBERRA"
90 LETA$(25)="TASMANIA"
95 LETA$(26)="DARWIN"
96 GOTO5000
100 CLS:PRINT@12,"[redacted]":PRINT@42,"[redacted]"
110 PRINT@71,"[redacted]":PRINT@101,"[redacted]"
120 PRINT@133,"[redacted]"
130 PRINT@163,"[redacted]"
140 PRINT@193,"[redacted]"
150 PRINT@224,"[redacted]"
160 PRINT@256,"[redacted]"
170 PRINT@288,"[redacted]"
180 PRINT@321,"[redacted]"
190 PRINT@354,"[redacted]"
200 PRINT@387,"[redacted]"
210 PRINT@419,"[redacted]":PRINT@435,"[redacted]":PRINT@469,"[redacted]"
220 PRINT@502,"[redacted]"
230 FORQ=1TO5000:NEXT
240 GOTO5100
500 PRINT@106,A$(2):PRINT@138,A$(12):RETURN
510 PRINT@179,A$(13):PRINT@212,A$(15):RETURN
520 PRINT@275,A$(17);" ";A$(19):PRINT@309,A$(21):RETURN
530 PRINT@195,A$(7):PRINT@226,A$(1):RETURN
540 PRINT@237,A$(19):PRINT@269,A$(28):RETURN
550 PRINT@375,A$(14):RETURN
560 PRINT@437,A$(23):RETURN
570 PRINT@494,A$(25):RETURN
1000 PRINT@219,LEFT$(A$(18),4):PRINT@251,RIGHT$(A$(18),4)
1005 RETURN
1010 PRINT@378,A$(20):RETURN
1020 PRINT@408,A$(24):RETURN
1030 PRINT@439,A$(22):RETURN
1040 PRINT@504,A$(3):RETURN
1050 PRINT@332,A$(4):RETURN
1060 PRINT@386,A$(16):RETURN
1070 PRINT@14,A$(26):RETURN
2000 PRINT@5,A$(11):PRINT@38,A$(10):RETURN
2010 PRINT@96,A$(8):PRINT@128,A$(5):RETURN
2020 PRINT@57,A$(6):PRINT@90,A$(5):RETURN
6232 IFLEFT$(X$,6)=A$(20)THENGOSUB1010
6234 IFLEFT$(X$,8)=A$(24)THENGOSUB1020
6236 IFLEFT$(X$,9)=A$(22)THENGOSUB1030
6238 IFLEFT$(X$,6)=A$(3)THENGOSUB1040
6240 IFLEFT$(X$,8)=A$(4)THENGOSUB1050
6242 IFLEFT$(X$,5)=A$(16)THENGOSUB1060
6244 IFLEFT$(X$,6)=A$(26)THENGOSUB1070
6250 NEXT
6261 PRINT@392,"    ":PRINT@422,"                    "
6262 PRINT@448,"                                    "
6265 FORQ=1TO9999:NEXT
6270 GOTO5000
6300 FORI=1TO5
6309 PRINT@448,"                                    "
6310 IFI=1THENPRINT@8,"!";
6311 IFI=2THENPRINT@98,"!";
6312 IFI=3THENPRINT@92,"!";
6313 IFI=4THENPRINT@446,"!";
6314 IFI=5THENPRINT@492,"!";
6320 PRINT@395,"NAME":PRINT@422,"THE OCEAN?":PRINT:INPUTX$
6325 COLOR7,1
6330 IFLEFT$(X$,9)="TIMOR SEA"THENGOSUB2000
6332 IFLEFT$(X$,12)="INDIAN OCEAN"THENGOSUB2010
6334 IFLEFT$(X$,13)="PACIFIC OCEAN"THENGOSUB2020
6336 IFLEFT$(X$,10)="TASMAN SEA"THENGOSUB2030
6338 IFLEFT$(X$,14)="SOUTHERN OCEAN"THENGOSUB2040
6350 NEXT
6360 GOTO 6265
```

Disk Drive

Dick Smith Electronics is thinking about an 80 K floppy disk drive for the VZ 200.

The project is at an early stage at present but DSE would like to hear from people who would buy a disk drive if one was available.

The price is expected to be about $300, but remember this is a preliminary estimate only.

If you are interested let DSE head office know.

```
2030 PRINT@410,A$(9):PRINT@444,A$(10):RETURN
2040 PRINT@395,"    ":PRINT@422,"                    "
2041 PRINT@448,"                                    ":PRINT@492," "
2042 PRINT@425,A$(27):PRINT@460,A$(5):RETURN
5000 COLOR,0
5010 CLS:PRINT@42,"GEOGRAPHY MENU":PRINT@74,"**************"
5020 PRINT@106,"1.  MAP":PRINT@138,"2.  COUNTRY"
5030 PRINT@170,"3.  STATES":PRINT@202,"4.  CAPITALS"
5040 PRINT@234,"5.  OCEANS":PRINT
5042 PRINT@266,"6.  STOP"
5045 INPUTZ
5050 IFZ=1THENGOTO100
5060 IFZ=2THENGOSUB100
5070 IFZ=3THENGOSUB100
5080 IFZ=4THENGOSUB100
5090 IFZ=5THENGOSUB100
5095 IFZ=6THENEND
5100 IFZ=2THENGOTO6000
5110 IFZ=3THENGOTO6100
5120 IFZ=4THENGOTO6200
5130 IFZ=5THENGOTO6300
5140 GOTO5000
6000 PRINT@395,"NAME":PRINT@422,"THE COUNTRY?":PRINT:INPUTX$
6010 IFLEFT$(X$,9)<>A$(1)THENGOTO6010
6014 PRINT@448,"                                    "
6015 PRINT@395,"    ":PRINT@422,"                    "
6016 COLOR3,1:PRINT@202,A$(1);
6020 GOTO6265
6100 FORI=2TO16STEP2
6101 IFI=2THENPRINT@112,"#";
6102 IFI=4THENPRINT@181,"#";
6103 IFI=6THENPRINT@276,"#";
6104 IFI=8THENPRINT@200,"#";
6105 IFI=10THENPRINT@239,"#";
6106 IFI=12THENPRINT@378,"#";
6107 IFI=14THENPRINT@439,"#";
6108 IFI=16THENPRINT@501,"#";
6110 PRINT@448,"                                    "
6120 PRINT@395,"NAME":PRINT@422,"THE STATE?":PRINT:INPUTX$
6125 COLOR2,1
6130 IFLEFT$(X$,18)="NORTHERN TERRITORY"THENGOSUB500
6132 IFLEFT$(X$,10)="QUEENSLAND"THENGOSUB510
6134 IFLEFT$(X$,15)="NEW SOUTH WALES"THENGOSUB520
6136 IFLEFT$(X$,17)="WESTERN AUSTRALIA"THENGOSUB530
6138 IFLEFT$(X$,15)="SOUTH AUSTRALIA"THENGOSUB540
6140 IFLEFT$(X$,6)="A.C.T."THENGOSUB550
6142 IFLEFT$(X$,8)=A$(23)THENGOSUB560
6144 IFLEFT$(X$,8)=A$(25)THENGOSUB570
6160 NEXT
6161 PRINT@395,"    ":PRINT@422,"                    "
6162 PRINT@448,"                                    "
6165 GOTO6265
6200 FORI=1TO8
6209 PRINT@448,"                                    "
6210 IFI=1THENPRINT@219,"*";
6211 IFI=2THENPRINT@378,"*";
6212 IFI=3THENPRINT@408,"*";
6213 IFI=4THENPRINT@439,"*";
6214 IFI=5THENPRINT@504,"*";
6215 IFI=6THENPRINT@370,"*";
6216 IFI=7THENPRINT@387,"*";
6218 IFI=8THENPRINT@14,"*";
6220 PRINT@395,"NAME":PRINT@422,"THE CAPITAL?":PRINT:INPUTX$
6225 COLOR4,1
6230 IFLEFT$(X$,8)="BRISBANE"THENGOSUB1000
```